Invoke

Release

Nov 12, 2018

Contents

This website covers project information for Invoke such as the changelog, contribution guidelines, development roadmap, news/blog, and so forth. Detailed usage and API documentation can be found at our code documentation site, docs.pyinvoke.org.

Please see below for a high level intro, or the navigation on the left for the rest of the site content.

CHAPTER 1

What is Invoke?

Invoke is a Python (2.7 and 3.4+) task execution tool & library, drawing inspiration from various sources to arrive at a powerful & clean feature set.

- Like Ruby's Rake tool and Invoke's own predecessor Fabric 1.x, it provides a clean, high level API for running shell commands and defining/organizing task functions from a `tasks.py` file:

```
from invoke import task

@task
def clean(c, docs=False, bytecode=False, extra=''):
    patterns = ['build']
    if docs:
        patterns.append('docs/_build')
    if bytecode:
        patterns.append('**/*.pyc')
    if extra:
        patterns.append(extra)
    for pattern in patterns:
        c.run("rm -rf {}".format(pattern))

@task
def build(c, docs=False):
    c.run("python setup.py build")
    if docs:
        c.run("sphinx-build docs docs/_build")
```

- From GNU Make, it inherits an emphasis on minimal boilerplate for common patterns and the ability to run multiple tasks in a single invocation:

```
$ invoke clean build
```

- Where Fabric 1.x considered the command-line approach the default mode of use, Invoke (and tools built on it) are equally at home embedded in your own Python code or a REPL:

```
>>> from invoke import run
>>> cmd = "pip install -r requirements.txt"
>>> result = run(cmd, hide=True, warn=True)
>>> print(result.ok)
True
>>> print(result.stdout.splitlines()[-1])
Successfully installed invocations-0.13.0 pep8-1.5.7 spec-1.3.1
```

- Following the lead of most Unix CLI applications, it offers a traditional flag-based style of command-line parsing, deriving flag names and value types from task signatures (optionally, of course!):

```
$ invoke clean --docs --bytecode build --docs --extra='**/*.pyo'
$ invoke clean -d -b build --docs -e '**/*.pyo'
$ invoke clean -db build -de '**/*.pyo'
```

- Like many of its predecessors, it offers advanced features as well – namespacing, task aliasing, before/after hooks, parallel execution and more.

1.1 Changelog

- : Allow the configuration system to override which `Executor` subclass to use when executing tasks (via an import-oriented string).

  Specifically, it's now possible to alter execution by distributing such a subclass alongside, for example, a repository-local config file which sets `tasks.executor_class`; previously, this sort of thing required use of custom binaries.

- #466: Update the parsing and CLI-program mechanisms so that all core arguments may be given within task CLI contexts; previously this functionality only worked for the `--help` flag, and other core arguments given after task names (such as `--echo`) were silently ignored.

- #301: (via #414) Overhaul tab completion mechanisms so users can print a completion script which automatically matches the emitting binary's configured names (compared to the previous hardcoded scripts, which only worked for `inv`/`invoke` by default). Thanks to Nicolas Höning for the foundational patchset.

- #559: (also fabric/fabric#1812) Modify how `Runner` performs stdin terminal mode changes, to avoid incorrect terminal state restoration when run concurrently (which could lead to things like terminal echo becoming disabled after the Python process exits).

  Thanks to Adam Jensen and Nick Timkovich for the detailed bug reports & reproduction assistance.

- #556: (also fabric/fabric#1823) Pre-emptively check for an error condition involving an unpicklable config file value (Python config files and imported module objects) and raise a useful exception instead of allowing a confusing `TypeError` to bubble up later. Reported by Pham Cong Dinh.

- #559: (also fabric/fabric#1812) Modify how `Runner` performs stdin terminal mode changes, to avoid incorrect terminal state restoration when run concurrently (which could lead to things like terminal echo becoming disabled after the Python process exits).

  Thanks to Adam Jensen and Nick Timkovich for the detailed bug reports & reproduction assistance.

- #556: (also fabric/fabric#1823) Pre-emptively check for an error condition involving an unpicklable config file value (Python config files and imported module objects) and raise a useful exception instead of allowing a confusing `TypeError` to bubble up later. Reported by Pham Cong Dinh.

- #543: Implemented support for using `INVOKE_RUNTIME_CONFIG` env var as an alternate method of supplying a runtime configuration file path (effectively, an env var based version of using the `-f`/`--config` option). Feature request via Kevin J. Qiu.

- : Add a `klass` kwarg to `@task` to allow extending codebases the ability to create their own variants on `@task`/`Task`.
- : Remove overzealous argument checking in `@task`, instead just handing any extra kwargs into the task class constructor. The high level behavior for truly invalid kwargs is the same (`TypeError`) but now extending codebases can add kwargs to their versions of `@task` without issue.
- : Refactor `Call` internals slightly, exposing some previously internal logic as the `clone_data` method; this is useful for client codebases when extending `Call` and friends.
- : Enhance `Call` with a new method (`clone_data`) and new kwarg to an existing method (`clone` grew `with_`) to assist subclassers when extending.
- : Implemented some minor missing tests, such as testing the `INVOKE_DEBUG` low-level env var.
- : Fix some test-suite-only failures preventing successful testing on Python 3.7 and PyPy3, and move them out of the 'allowed failures' test matrix quarantine now that they pass.
- : Apply the black code formatter to our codebase and our CI configuration.
- : Fixed some inaccuracies in the API docs around `Executor` and its `core` kwarg (was erroneously referring to `ParserContext` instead of `ParseResult`). Includes related cleaning-up of docstrings and tests.
- : As part of solving #528 we found a related bug, where unnamed subcollections also caused issues with `inv --list --list-format=json`. Specifically, `Collection.serialized` sorts subcollections by name, which is problematic when that name is `None`. This is now fixed.
- #528: Around Invoke 0.23 we broke the ability to weave in subcollections via keyword arguments to `Collection`, though it primarily manifests as `NoneType` related errors during `inv --list`. This was unintentional and has been fixed. Report submitted by Tuukka Mustonen.
- : Fix up the `__repr__` of `Call` to reference dynamic class name instead of hardcoding `"Call"`; this allows subclasses' `__repr__` output to be correct instead of confusing.
- #270: (also #551) `None` values in config levels (most commonly caused by empty configuration files) would raise `AttributeError` when `merge_dicts` was used to merge config levels together. This has been fixed. Thanks to Tyler Hoffman and Vlad Frolov for the reports.
- : Implemented some minor missing tests, such as testing the `INVOKE_DEBUG` low-level env var.
- : Fix some test-suite-only failures preventing successful testing on Python 3.7 and PyPy3, and move them out of the 'allowed failures' test matrix quarantine now that they pass.
- : Apply the black code formatter to our codebase and our CI configuration.
- : Fixed some inaccuracies in the API docs around `Executor` and its `core` kwarg (was erroneously referring to `ParserContext` instead of `ParseResult`). Includes related cleaning-up of docstrings and tests.
- : Updated `Task` to mimic the wrapped function's `__module__` attribute, allowing for better interaction with things like Sphinx autodoc that attempt to filter out imported objects from a module.
- : Added the –prompt-for-sudo-password CLI option for getpass-based up-front prompting of a sensitive configuration value.
- : Tweaked the innards of `Config`/`DataProxy` to prevent accessing properties & other attributes' values during `__setattr__` (the code in question only needed the names). This should have no noticeable effect on user code (besides a marginal speed increase) but fixed some minor test coverage issues.
- : Removed an old, unused and untested (but, regrettably, documented and public) method that doesn't seem to be much use: `invoke.config.Config.paths`. Please reach out if you were actually using it and we may consider adding some form of it back.

> **Warning:** This is a backwards incompatible change if you were using `Config.paths`.

- #33: Overhaul task listing (formerly just a simple, boolean `--list`) to make life easier for users with nontrivial task trees:
  - Limit display to a specific namespace by giving an optional argument to `--list`, e.g. `--list build`;
  - Additional output formats besides the default (now known as `flat`) such as a nested view with `--list-format nested` or script-friendly output with `--list-format json`.
  - The default `flat` format now sorts a bit differently - the previous algorithm would break up trees of tasks.
  - Limit listing depth, so it's easier to view only the first level or two (i.e. the overall namespaces) of a large tree, e.g. `--list --list-depth 1`;

  Thanks to the many users who submitted various requests under this ticket's umbrella, and to Dave Burkholder in particular for detailed use case analysis & feedback.
- : (partially re: #449) Update error message around missing positional arguments so it actually lists them. Includes a minor tweak to the API of `ParserContext`, namely changing `needs_positional_arguments` (bool) to `missing_positional_arguments` (list).
- #516: Remove the CLI parser ambiguity rule regarding flag-like tokens which are seen after an optional-value flag (e.g. `inv task --optionally-takes-a-value --some-other-flag`.) Previously, any flag-like value in such a spot was considered ambiguous and raised a `ParseError`. Now, the surrounding parse context is used to resolve the ambiguity, and no error is raised.

> **Warning:** This behavior is backwards incompatible, but only if you had the minority case where users frequently *and erroneously* give otherwise-legitimate flag-like values to optional-value arguments, and you rely on the parse errors to notify them of their mistake. (If you don't understand what this means, don't worry, you almost certainly don't need to care!)

- : Integer-type CLI arguments were not displaying placeholder text in `--help` output (i.e. they appeared as `--myint` instead of `--myint=INT`.) This has been fixed.
- : `Collection` had some minor bugs or oversights in how it responds to things like `repr()`, `==`; boolean behavior; how docstrings appear when created from a Python module; etc. All are now fixed. If you're not sure whether this affects you, it does not :)
- : Previously, some error conditions (such as invalid task or collection names being supplied by the user) printed to standard output, instead of standard error. Standard error seems more appropriate here, so this has been fixed.

> **Warning:** This is backwards incompatible if you were explicitly checking the standard output of the `inv[oke]` program for some of these error messages.

> **Warning:** If your code is manually raising or introspecting instances of `Exit`, note that its signature has changed from `Exit(code=0)` to `Exit(message=None, code=None)`. (Thus, this will only impact you if you were calling its constructor instead of raising the class object itself.)

- #469: Fix up the doc/example re: subclassing `Config`. Credit: `@Aiky30`.
- #433: Add -dev and -nightly style Python versions to our Travis builds. Thanks to `@SylvainDe` for the contribution.

- : Rename `invoke.platform` to `invoke.terminals`; it was inadvertently shadowing the `platform` standard library builtin module. This was not causing any bugs we are aware of, but it is still poor hygiene.

  > **Warning:** This change is technically backwards incompatible. We don't expect many users import `invoke.platform` directly, but if you are, take note.

- #515: Ported the test suite from spec (nose) to pytest-relaxed (pytest) as pytest basically won the test-runner war against nose & has greater mindshare, more shiny toys, etc.
- : (partially re: #33) Renamed the `--root` CLI flag to `--search-root`, partly for clarity (#33 will be adding namespace display-root related flags, which would make `--root` ambiguous) and partly for consistency with the config option, which was already named `search_root`. (The short version of the flag, `-r`, is unchanged.)

  > **Warning:** This is a backwards incompatible change. To fix, simply use `--search-root` anywhere you were previously using `--root`.

- #488: Account for additional I/O related `OSError` error strings when attempting to capture only this specific subtype of error. This should fix some issues with less common libc implementations such as `musl` (as found on e.g. Alpine Linux.) Thanks to Rajitha Perera for the report.
- #437: When merging configuration levels together (which uses `copy.copy` by default), pass file objects by reference so they don't get closed. Catch & patch by Paul Healy.
- #342: Accidentally hardcoded `Collection` instead of `cls` in `Collection.from_module` (an alternate constructor and therefore a classmethod.) This made it rather hard to properly subclass `Collection`. Report and initial patch courtesy of Luc Saffre.
- #469: Fix up the doc/example re: subclassing `Config`. Credit: `@Aiky30`.
- #433: Add -dev and -nightly style Python versions to our Travis builds. Thanks to `@SylvainDe` for the contribution.
- : Iterable-type CLI args were actually still somewhat broken & were 'eating' values after themselves in the parser stream (thus e.g. preventing parsing of subsequent tasks or flags.) This has been fixed.
- #407: (also #494, #67) Update the default value of the `run.shell` config value so that it reflects a Windows-appropriate value (specifically, the `COMSPEC` env var or a fallback of `cmd.exe`) on Windows platforms. This prevents Windows users from being forced to always ship around configuration-level overrides.

  Thanks to Maciej 'maQ' Kusz for the original patchset, and to `@thebjorn` and Garrett Jenkins for providing lots of feedback.
- #364: Drop Python 2.6 and Python 3.3 support, as these versions now account for only very low percentages of the userbase and are unsupported (or about to be unsupported) by the rest of the ecosystem, including `pip`.

  This includes updating documentation & packaging metadata as well as taking advantage of basic syntax additions like set literals/comprehensions (`{1, 2, 3}` instead of `set([1, 2, 3])`) and removing positional string argument specifiers (`"{}".format(val)` instead of `"{0}".format(val)`).
- #132: Implement 'iterable' and 'incrementable' CLI flags, allowing for invocations like `inv mytask --listy foo --listy bar` (resulting in a call like `mytask(listy=['foo', 'bar'])`) or `inv mytask -vvv` (resulting in e.g. `mytask(verbose=3)`. Specifically, these require use of the new iterable and incrementable arguments to `@task` - see those links to the conceptual docs for details.
- : The behavior of `Config` when `lazy=True` didn't match that described in the API docs, after the recent updates to its lifecycle. (Specifically, any config data given to the constructor was not visible in the resulting instance until `merge()` was explicitly called.) This has been fixed, along with other related minor issues.

- #467: (Arguably also a feature, but since it enables behavior users clearly found intuitive, we're considering it a bug.) Split up the parsing machinery of `Program` and pushed the `Collection`-making out of `Loader`. Combined, this allows us to honor the project-level config file *before* the second (task-oriented) CLI parsing step, instead of after.

  For example, this means you can turn off `auto_dash_names` in your per-project configs and not only in your system or user configs.

  Report again courtesy of Luke Orland.

  > **Warning:** This is a backwards incompatible change *if* you were subclassing and overriding any of the affected methods in the `Program` or `Loader` classes.

- #465: The `tasks.auto_dash_names` config option added in `0.20.0` wasn't being fully honored when set to `False`; this has been fixed. Thanks to Luke Orland for the report.
- : Fix a broken `six.moves` import within `invoke.util`; was causing `ImportError` in environments without an external copy of `six` installed.

  The dangers of one's local and CI environments all pulling down packages that use `six`! It's everywhere!
- #322: Allow users to completely disable mirroring of stdin to subprocesses, by specifying `False` for the `run.in_stream` config setting and/or keyword argument.

  This can help prevent problems when running Invoke under systems that have no useful standard input and which otherwise defeat our pty/fileno related detection.
- #329: All task and collection names now have underscores turned into dashes automatically, as task parameters have been for some time. This impacts `--list`, `--help`, and of course the parser. For details, see Dashes vs underscores.

  This behavior is controlled by a new config setting, `tasks.auto_dash_names`, which can be set to `False` to go back to the classic behavior.

  Thanks to Alexander Artemenko for the initial feature request.
- #310: (also #455, #291) Allow configuring collection root directory & module name via configuration files (previously, they were only configurable via CLI flags or generating a custom `Program`.)
- : (required to support #310 and #329) Break up the `Config` lifecycle some more, allowing it to gradually load configuration vectors; this allows the CLI machinery (`Executor`) to honor configuration settings from config files which impact how CLI parsing and task loading behaves.

  Specifically, this adds more public `Config.load_*` methods, which in tandem with the `lazy` kwarg to `__init__` (formerly `defer_post_init`, see below) allow full control over exactly when each config level is loaded.

  > **Warning:** This change may be backwards incompatible if you were using or subclassing the `Config` class in any of the following ways:
  >
  > – If you were passing `__init__` kwargs such as `project_home` or `runtime_path` and expecting those files to auto-load, they no longer do; you must explicitly call `load_project` and/or `load_runtime` explicitly.
  >
  > – The `defer_post_init` keyword argument to `Config.__init__` has been renamed to `lazy`, and controls whether system/user config files are auto-loaded.
  >
  > – `Config.post_init` has been removed, in favor of explicit/granular use of the `load_*` family of methods.

> – All `load_*` methods now call `Config.merge` automatically by default (previously, merging was deferred to the end of most config related workflows.)
>
> This should only be a problem if your config contents are extremely large (it's an entirely in-memory dict-traversal operation) and can be avoided by specifying `merge=False` to any such method. (Note that you must, at some point, call `merge` in order for the config object to work normally!)

- : Display of hidden subprocess output when a command execution failed (end-of-session output starting with `Encountered a bad command exit code!`) was liable to display encoding errors (e.g. `'ascii' codec can't encode character ...`) when that output was not ASCII-compatible.

  This problem was previously solved for *non-hidden* (mirrored) subprocess output, but the fix (encode the data with the local encoding) had not been applied to exception display. Now it's applied in both cases.

- #396: `Collection.add_task(task, aliases=('other', 'names')` was listed in the conceptual documentation, but not implemented (technically, it was removed at some point and never reinstated.) It has been (re-)added and now exists. Thanks to `@jenisys` for the report.

  > **Warning:** This technically changes argument order for `Collection.add_task`, so be aware if you were using positional arguments!

- : Add `MockContext.set_result_for` to allow massaging a mock Context's configured results after instantiation.

- : Update Context internals re: command execution & configuration of runner subclasses, to work better in client libraries such as Fabric 2.

  > **Note:** If you were using the undocumented `runner` configuration value added in #446, it is now `runners.local`.

  > **Warning:** This change modifies the internals of methods like `run` and `sudo`; users maintaining their own subclasses should be aware of possible breakage.

- #444: Add support for being used as `python -m invoke <args>` on Python 2.7 and up. Thanks to Pekka Klärck for the feature request.

- #205: Allow giving core flags like `--help` after tasks to trigger per-task help. Previously, only `inv --help taskname` worked.

  > **Note:** Tasks with their own `--help` flags won't be able to leverage this feature - the parser will still interpret the flag as being per-task and not global. This may change in the future to simply throw an exception complaining about the ambiguity. (Feedback welcome.)

- #446: Implement `cd` and `prefix` context managers (as methods on the not-that-one-the-other-one `Context` class.) These are based on similar functionality in Fabric 1.x. Credit: Ryan P Kilby.

- #448: Fix up some config-related tests that have been failing on Windows for some time. Thanks to Ryan P Kilby.

- : Add a `user` kwarg & config parameter to `Context.sudo`, which corresponds roughly to `sudo -u <user> <command>`.

- #440: Make sure to skip a call to `struct/ioctl` on Windows platforms; otherwise certain situations inside `run` calls would trigger import errors. Thanks to `@chrisc11` for the report.
- #425: Fix `Inappropriate ioctl for device` errors (usually `OSError`) when running Invoke without a tty-attached stdin (i.e. when run under 'headless' continuous integration systems or simply as e.g. `inv sometask < /dev/null` (redirected stdin.) Thanks to Javier Domingo Cansino for the report & Tuukka Mustonen for troubleshooting assistance.
- #439: Avoid placing stdin into bytewise read mode when it looks like Invoke has been placed in the background by a shell's job control system; doing so was causing the shell to pause the Invoke process (e.g. with a message like `suspended (tty output)`.) Reported by Tuukka Mustonen.
- : Even more setup.py related tomfoolery.
- : Deal with the fact that PyPI's rendering of Restructured Text has no idea about our fancy new use of Sphinx's doctest module. Sob.
- : Fix a silly typo preventing proper rendering of the packaging `long_description` (causing an effectively blank PyPI description.)
- : `Result` and `UnexpectedExit` objects now have a more useful `repr()` (and in the case of `UnexpectedExit`, a distinct `repr()` from their preexisting `str()`.)
- : `Context.sudo` no longer prompts the user when the configured sudo password is empty; thus, an empty sudo password and a `sudo` program configured to require one will result in an exception.

  The runtime prompting for a missing password was a temporary holdover from Fabric v1, and in retrospect is undesirable. We may add it back in as an opt-in behavior (probably via subclassing) in the future if anybody misses it.

  > **Warning: This is a backwards incompatible change**, if you were relying on `sudo()` prompting you for your password (vs configuring it). If you *were* doing that, you can simply switch to `run("sudo <command>")` and respond to the subprocess' sudo prompt by hand instead.

- : Switched the order of the first two arguments of `Config.__init__`, so that the `overrides` kwarg becomes the first positional argument.

  This supports the common use case of making a `Config` object that honors the system's core/global defaults; previously, because `defaults` was the first argument, you'd end up replacing those core defaults instead of merging with them.

  > **Warning: This is a backwards incompatible change** if you were creating custom `Config` objects via positional, instead of keyword, arguments. It should have no effect otherwise.

- #309: Overhaul how task execution contexts/configs are handled, such that all contexts in a session now share the same config object, and thus user modifications are preserved between tasks. This has been done in a manner that should not break things like collection-based config (which may still differ from task to task.)

  > **Warning: This is a backwards incompatible change** if you were relying on the post-0.12 behavior of cloning config objects between each task execution. Make sure to investigate if you find tasks affecting one another in unexpected ways!

- #418: Enhance ability of client libraries to override config filename prefixes. This includes modifications to related functionality, such as how env var prefixes are configured.

Warning: This is a backwards incompatible change if:

- you were relying on the `env_prefix` keyword argument to `Config.__init__`; it is now the `prefix` or `env_prefix` class attribute, depending.
- or the kwarg/attribute of the same name in `Program.__init__`; you should now be subclassing `Config` and using its `env_prefix` attribute;
- or if you were relying on how standalone `Config` objects defaulted to having a `None` value for `env_prefix`, and thus loaded env vars without an `INVOKE_` style prefix.

  See new documentation for this functionality at Customizing the configuration system's defaults for details.

- #232: Add support for `.yml`-suffixed config files (in addition to `.yaml`, `.json` and `.py`.) Thanks to Matthias Lehmann for the original request & Greg Back for an early patch.
- #430: Fallback importing of PyYAML when Invoke has been installed without its vendor directory, was still trying to import the vendorized module names (e.g. `yaml2` or `yaml3` instead of simply `yaml`). This has been fixed, thanks to Athmane Madjoudj.
- #432: Tighten application of IO thread `join` timeouts (in `run`) to only happen when #351 appears actually present. Otherwise, slow/overworked IO threads had a chance of being joined before truly reading all data from the subprocess' pipe.
- : Fixed some Python 2.6 incompatible string formatting that snuck in recently.
- : `Config.clone` grew a new `into` kwarg allowing client libraries with their own `Config` subclasses to easily "upgrade" vanilla Invoke config objects into their local variety.
- #421: Updated `Config.clone` (and a few other related areas) to replace use of `copy.deepcopy` with a less-rigorous but also less-likely-to-explode recursive dict copier. This prevents frustrating `TypeErrors` while still preserving barriers between different tasks' configuration values.
- : `Config`'s internals got cleaned up somewhat; end users should not see much of a difference, but advanced users or authors of extension code may notice the following:
  - Direct modification of config data (e.g. `myconfig.section.subsection.key = 'value'` in user/task code) is now stored in its own config 'level'/data structure; previously such modifications simply mutated the central, 'merged' config cache. This makes it much easier to determine where a final observed value came from, and prevents accidental data loss.
  - Ditto for deleted values.
  - Merging/reconciliation of the config levels now happens automatically when data is loaded or modified, which not only simplifies the object's lifecycle a bit but allows the previous change to function without requiring users to call `.merge()` after every modification.
- #413: Update behavior of `DataProxy` (used within `Context` and `Config`) again, fixing two related issues:
  - Creating new configuration keys via attribute access wasn't possible: one had to do `config['foo'] = 'bar'` because `config.foo = 'bar'` would set a real attribute instead of touching configuration.
  - Supertypes' attributes weren't being considered during the "is this a real attribute on `self`?" test, leading to different behavior between a nested config-value-as-attribute and a top-level Context/Config one.
- : Fix configuration framework such that nested or dict-like config values may be compared with regular dicts. Previously, doing so caused an `AttributeError` (as regular dicts lack a `.config`).
- #419: Optional parser arguments had a few issues:

  - The conceptual docs about CLI parsing mentioned them, but didn't actually show via example how to enable the feature, implying (incorrectly) that they were active always by default. An example has been added.
  - Even when enabled, they did not function correctly when their default values were of type `bool`; in this situation, trying to give a value (vs just giving the flag name by itself) caused a parser error. This has been fixed.

  Thanks to `@ouroboroscoding` for the report.

- : Configuration keys named `config` were inadvertently exposing the internal dict representation of the containing config object, instead of displaying the actual value stored in that key. (Thus, a set config of `mycontext.foo.bar.config` would act as if it was the key/value contents of the `mycontext.foo.bar` subtree.) This has been fixed.
- : Python 3's hashing rules differ from Python 2, specifically:

  > A class that overrides `__eq__()` and does not define `__hash__()` will have its `__hash__()` implicitly set to None.

  `Config` (specifically, its foundational class `DataProxy`) only defined `__eq__` which, combined with the above behavior, meant that `Config` objects appeared to hash successfully on Python 2 but yielded `TypeErrors` on Python 3.

  This has been fixed by explicitly setting `__hash__ = None` so that the objects do not hash on either interpreter (there are no good immutable attributes by which to define hashability).
- #426: `DataProxy` based classes like `Config` and `Context` didn't like being `pickled` or `copied` and threw `RecursionError`. This has been fixed.
- #204: (via #412) Fall back to globally-installed copies of our vendored dependencies, if the import from the `vendor` tree fails. In normal situations this won't happen, but it allows advanced users or downstream maintainers to nuke `vendor/` and prefer explicitly installed packages of e.g. `six`, `pyyaml` or `fluidity`. Thanks to Athmane Madjoudj for the patch.
- #369: Overhaul the autoresponse functionality for `run` so it's significantly more extensible, both for its own sake and as part of implementing #294 (see its own changelog entry for details).

  Warning: This is a backwards incompatible change: the `responses` kwarg to `run()` is now `watchers`, and accepts a list of `StreamWatcher` objects (such as `Responder`) instead of a dict.

  If you were using `run(..., responses={'pattern': 'response'}` previously, just update to instead use `run(..., watchers=[Responder('pattern', 'response')])`.

- #294: Implement `Context.sudo`, which wraps `run` inside a `sudo` command. It is capable of auto-responding to `sudo`'s password prompt with a configured password, and raises a specific exception (`AuthFailure`) if that password is rejected.
- : Update implementation of `Result` so it has default values for all parameters/attributes. This allows it to be more easily used when mocking `run` calls in client libraries' tests.

  Warning: This is a backwards incompatible change if you are manually instantiating `Result` objects with positional arguments: positional argument order has changed. (Compare the API docs between versions to see exactly how.)

- : Add a `MockContext` class for easier testing of user-written tasks and related client code. Includes adding a conceptual document on how to test Invoke-using code.

- #406: Update handling of Ctrl-C/`KeyboardInterrupt`, and subprocess exit status pass-through, to be more correct than before:
  - Submit the interrupt byte sequence `\x03` to stdin of all subprocesses, instead of sending `SIGINT`.
    - This results in behavior closer to that of truly pressing Ctrl-C when running subprocesses directly; for example, interactive programs like `vim` or `python` now behave normally instead of prematurely exiting.
    - Of course, programs that would normally exit on Ctrl-C will still do so!
  - The exit statuses of subprocesses run with `pty=True` are more rigorously checked (using `os.WIFEXITED` and friends), allowing us to surface the real exit values of interrupted programs instead of manually assuming exit code `130`.
    - Typically, this will be exit code `-2`, but it is system dependent.
    - Other, non-Ctrl-C-driven signal-related exits under PTYs should behave better now as well - previously they could appear to exit `0`!
  - Non-subprocess-related `KeyboardInterrupt` (i.e. those generated when running top level Python code outside of any `run` function calls) will now trigger exit code `1`, as that is how the Python interpreter typically behaves if you `KeyboardInterrupt` it outside of a live REPL.

  Warning: These changes are **backwards incompatible** if you were relying on the "exits `130`" behavior added in version 0.13, or on the (incorrect) `SIGINT` method of killing pty-driven subprocesses on Ctrl-C.

- : Expose the (normalized) value of `run`'s `hide` parameter in its return-value `Result` objects.
- : Fix a bug in `Config.clone` where it was instantiating a new `Config` instead of a member of the subclass.
- : Correctly raise `TypeError` when unexpected keyword arguments are given to `run`.
- #58: Work around bugs in `select()` when handling subprocess stream reads, which was causing poor behavior in many nontrivial interactive programs (such as `vim` and other fullscreen editors, `python` and other REPLs/shells, etc). Such programs should now be largely indistinguishable from their behavior when run directly from a user's shell.
- : Fix `DataProxy` (used within `Context` and `Config`) so that real attributes and methods which are shadowed by configuration keys, aren't proxied to the config during regular attribute get/set. (Such config keys are thus required to be accessed via dict-style only, or (on `Context`) via the explicit `.config` attribute.)
- #283: Fix the concepts/library docs so the example of an explicit `namespace=` argument correctly shows wrapping an imported task module in a `Collection`. Thanks to `@zaiste` for the report.
- #288: Address a bug preventing reuse of Invoke as a custom binstub, by moving `--list` into the "core args" set of flags present on all Invoke-derived binstubs. Thanks to Jordon Mears for catch & patch.
- #349: Display the string representation of `UnexpectedExit` when handling it inside of `Program` (including regular `inv`), if any output was hidden during the `run` that generated it.

  Previously, we only exited with the exception's stored exit code, meaning failures of `run(..., hide=True)` commands were unexpectedly silent. (Library-style use of the codebase didn't have this problem, since tracebacks aren't muted.)

  While implementing this change, we also tweaked the overall display of `UnexpectedExit` so it's a bit more consistent & useful:

  - noting "hey, you ran with `pty=True`, so there's no stderr";
  - showing only the last 10 lines of captured output in the error message (users can, of course, always manually handle the error & access the full thing if desired);

  - only showing a given stream when it was not already printed to the user's terminal (i.e. if `hide=False`, no captured output is shown in the error text; if `hide='stdout'`, only stdout is shown in the error text; etc.)

  Thanks to Patrick Massot for the original bug report.

- #67: Added `shell` option to `run`, allowing control of the shell used when invoking commands. Previously, `pty=True` used `/bin/bash` and `pty=False` (the default) used `/bin/sh`; the new unified default value is `/bin/bash`.

  Thanks to Jochen Breuer for the report.

- #259: (also #280) Allow updating (or replacing) subprocess shell environments, via the `env` and `replace_env` kwargs to `run`. Thanks to Fotis Gimian for the report, `@philtay` for an early version of the final patch, and Erich Heine & Vlad Frolov for feedback.

- #114: Ripped off the band-aid and removed non-contextualized tasks as an option; all tasks must now be contextualized (defined as `def mytask(context, ...)` - see Defining and running task functions) even if not using the context. This simplifies the implementation as well as users' conceptual models. Thanks to Bay Grabowski for the patch.

  Warning: This is a backwards incompatible change!

- #152: (also #251, #331) Correctly handle `KeyboardInterrupt` during `run`, re: both mirroring the interrupt signal to the subprocess *and* capturing the local exception within Invoke's CLI handler (so there's no messy traceback, just exiting with code `130`).

  Thanks to Peter Darrow for the report, and to Mika Eloranta & Máté Farkas for early versions of the patchset.

- #351: Protect against `run` deadlocks involving exceptions in I/O threads & nontrivial amounts of unread data in the corresponding subprocess pipe(s). This situation should now always result in exceptions instead of hangs.

- #350: (also #274, #241, #262, #242, #321, #338) Clean up and reorganize encoding-related parts of the code to avoid some of the more common or egregious encode/decode errors surrounding clearly non-ASCII-compatible text. Bug reports, assistance, feedback and code examples courtesy of Paul Moore, Vlad Frolov, Christian Aichinger, Fotis Gimian, Daniel Nunes, and others.

- #314: (Partial fix.) Update `MANIFEST.in` so source distributions include some missing project-management files (e.g. our internal `tasks.py`). This makes unpacked sdists more useful for things like running the doc or build tasks.

- #319: Fixed an issue resulting from #255 which caused problems with how we generate release wheels (notably, some releases such as 0.12.1 fail when installing from wheels on Python 2).

  Note: As part of this fix, the next release will distribute individual Python 2 and Python 3 wheels instead of one 'universal' wheel. This change should be transparent to users.

  Thanks to `@ojos` for the initial report and Frazer McLean for some particularly useful feedback.

- #303: Make sure `run` waits for its IO worker threads to cleanly exit (such as allowing a `finally` block to revert TTY settings) when `KeyboardInterrupt` (or similar) aborts execution in the main thread. Thanks to Tony S Yu and Máté Farkas for the report.

- #314: (Partial fix.) Update `MANIFEST.in` so source distributions include some missing project-management files (e.g. our internal `tasks.py`). This makes unpacked sdists more useful for things like running the doc or build tasks.

- #289: Handful of issues, all fallout from #289, which failed to make it out the door for 0.12.0. More are on the way but these should address blockers for some users:
  - Windows support for the new stdin replication functionality (this was totally blocking Windows users, as reported in #302 - sorry!);
  - Stdin is now mirrored to stdout when no PTY is present, so you can see what you're typing (plus a new `run` option and config param, `echo_stdin`, allowing user override of this behavior);
  - Exposed the stdin read loop's sleep time as `Runner.input_sleep`;
  - Sped up some tests a bit.
- #305: (also #306) Fix up some test-suite issues causing failures on Windows/Appveyor. Thanks to Paul Moore.
- #308: Earlier changes to TTY detection & its use in determining features such as stdin pass-through, were insufficient to handle edge cases such as nested Invoke sessions or piped stdin to Invoke processes. This manifested as hangs and `OSError` messages about broken pipes.

  The issue has been fixed by overhauling all related code to use more specific and accurate checks (e.g. examining just `fileno` and/or just `isatty`).

  Thanks to Tuukka Mustonen and Máté Farkas for the report (and for enduring the subsequent flood of the project maintainer's stream-of-consciousness ticket updates).
- #173: Overhauled top level CLI functionality to allow reusing Invoke for distinct binaries, optionally with bundled task namespaces as subcommands. As a side effect, this functionality is now much more extensible to boot. Thanks to Erich Heine for feedback/suggestions during development.

  Warning: This change is backwards incompatible if you imported anything from the `invoke.cli` module (which is now rearchitected as `Program`). It should be transparent to everybody else.

  See also:

  Reusing Invoke's CLI module as a distinct binary
- #228: (partial) Modified and expanded implementation of `Executor`, `Task` and `Call` to make implementing task parameterization easier.
- #289: (also #263) Implement autoresponding for `run`.
- #68: Disable Python's bytecode caching by default, as it complicates our typical use case (frequently-changing .py files) and offers little benefit for human-facing startup times. Bytecode caching can be explicitly re-enabled by specifying `--write-pyc` at runtime. Thanks to Jochen Breuer for feature request and `@brutus` for initial patchset.
- : Fixed a bug in the parser where `invoke --takes-optional-arg avalue --anotherflag` was incorrectly considering `--anotherflag` to be an ambiguity error (as if `avalue` had not been given to `--takes-optional-arg`.
- #295: Make sure that `run`'s `hide=True` also disables echoing. Otherwise, "hidden" helper `run` calls will still pollute output when run as e.g. `invoke --echo ....`
- #296: Don't mutate `sys.path` on collection load if task's parent directory is already on `sys.path`.
- #297: Ignore leading and trailing underscores when turning task arguments into CLI flag names.
- #257: Fix a RecursionError under Python 3 due to lack of `__deepcopy__` on `Call` objects. Thanks to Markus Zapke-Gründemann for initial report and Máté Farkas for the patch.
- : Fix incorrect changelog URL in package metadata.

- : Removed the `-H` short flag, leaving just `--hide`. This was done to avoid conflicts with Fabric's host-oriented `-H` flag. Favoritism is real! Apologies.

  > **Warning:** This change is backwards compatible if you used `-H`.

- : Removed official Python 3.2 support; sibling projects also did this recently, it's simply not worth the annoyance given the userbase size.
- #144: Add code-coverage reporting to our CI builds (albeit CodeCov instead of coveralls.io). Includes rejiggering our project-specific coverage-generating tasks. Thanks to David Baumgold for the original request/PR and to Justin Abrahms for the tipoff re: CodeCov.
- #254: Add an `exclude` option in our `setup.py` so setuptools doesn't try loading our vendored PyYAML's Python 2 sub-package under Python 3 (or vice versa - though all reports were from Python 3 users). Thanks to `@yoshiya0503` for catch & initial patch.
- #265: Update our Travis config to select its newer build infrastructure and also run on PyPy3. Thanks to Omer Katz.
- : Fix incorrect changelog URL in package metadata.
- #235: Allow custom stream objects to be used in `run` calls, to be used instead of the defaults of `sys.stdout`/`sys.stderr`.

  > **Warning:** This change required a major cleanup/rearchitecture of the command execution implementation. The vendored `pexpect` module has been completely removed and the API of the `Runner` class has changed dramatically (though **the API for run() itself has not**).
  >
  > Be aware there may be edge-case terminal behaviors which have changed or broken as a result of removing `pexpect`. Please report these as bugs! We expect to crib small bits of what `pexpect` does but need concrete test cases first.

- : Detect local controlling terminal size (`pty_size`) and apply that information when creating pseudoterminals in `run` when `pty=True`.
- : Add a `.command` attribute to `Result` to preserve the command executed for post-execution introspection.
- #238: (partial fix) Update the `zsh` completion script to account for use of the `--collection` core flag.
- #239: Completion erroneously presented core flags instead of per-task flags when both are present in the invocation being completed (e.g. `inv --debug my_task -<tab>`). This has been fixed.
- #237: Completion output lacked "inverse" flag names (e.g. `--no-myoption` as a boolean negative version of a defaulting-to-True boolean `myoption`). This has been corrected.
- #234: (also #243) Preserve task-module load location when creating explicit collections with `from_module`; when this was not done, project-local config files were not loading correctly. Thanks to `@brutus` and Jan Willems for initial report & troubleshooting, and to Greg Back for identifying the fix.
- : Capture & reraise exceptions generated by command execution I/O threads, in the main thread, as a `ThreadException`.
- : Correctly handle situations where `sys.stdin` has been replaced with an object lacking `.fileno` (e.g., some advanced Python shells, headless code execution tools, etc). Previously, this situation resulted in an `AttributeError`.
- : Display stdout instead of stderr in the `repr()` of `Failure` objects, when a pseudo-terminal was used. Previously, failure display focused on the stderr stream, which is always empty under pseudo-terminals.

- : Tweak README to reflect recent(-ish) changes in `pip` re: users who install the development version via `pip` instead of using git.
- #224: Add a completion script for the `fish` shell, courtesy of Jaime Marquínez Ferrándiz.
- : Additional rearranging of `run`/`Runner` related concerns for improved subclassing, organization, and use in other libraries, including:
  - Changed the name of the `runner` module to `runners`.
  - Moved the top level `run` function from its original home in `invoke.runner` to `invoke.__init__`, to reflect the fact that it's now simply a convenience wrapper around `Runner`.
  - Tweaked the implementation of `Runner` so it can reference `Context` objects (useful for anticipated subclasses).

  Warning: These are backwards incompatible changes if your code was doing any imports from the `invoke.runner` module (including especially `invoke.runner.run`, which is now only `invoke.run`). Function signatures have **not** changed.

- : Tweak README to reflect recent(-ish) changes in `pip` re: users who install the development version via `pip` instead of using git.
- #147: Drastically overhaul/expand the configuration system to account for multiple configuration levels including (but not limited to) file paths, environment variables, and Python-level constructs (previously the only option). See Configuration for details. Thanks to Erich Heine for his copious feedback on this topic.

  Warning: This is technically a backwards incompatible change, though some existing user config-setting code may continue to work as-is. In addition, this system may see further updates before 1.0.

- #219: Fall back to non-PTY command execution in situations where `pty=True` but no PTY appears present. See `Local` for details.
- #104: Add core CLI flag `--complete` to support shell tab completion scripts, and add some 'blessed' such scripts for bash (3 and 4) and zsh. Thanks to Ivan Malison and Andrew Roberts for providing discussion & early patchsets.
- #175: `autoprint` did not function correctly for tasks stored in sub-collections; this has been fixed. Credit: Matthias Lehmann.
- #180: Empty invocation (e.g. just `invoke` with no flags or tasks, and when no default task is defined) no longer printed help output, instead complaining about the lack of default task. It now prints help again. Thanks to Brent O'Connor for the catch.
- #183: Task docstrings whose first line started on the same line as the opening quote(s) were incorrectly presented in `invoke --help <task>`. This has been fixed by using `inspect.getdoc`. Thanks to Pekka Klärck for the catch & suggested fix.
- #191: Bypass `pexpect`'s automatic command splitting to avoid issues running complex nested/quoted commands under a pty. Credit to `@mijikai` for noticing the problem.
- #201: (also #211) Replace the old, first-draft gross monkeypatched Popen code used for `invoke.runner.run` with a non-monkeypatched approach that works better on non-POSIX platforms like Windows, and also attempts to handle encoding and locale issues more gracefully (meaning: at all gracefully).

  Specifically, the new approach uses threading instead of `select.select`, and performs explicit encoding/decoding based on detected or explicitly expressed encodings.

Major thanks to Paul Moore for an enormous amount of testing/experimentation/discussion, as well as the bulk of the code changes themselves.

> **Warning:** The top level `invoke.runner.run` function has had a minor signature change: the sixth positional argument used to be `runner` and is now `encoding` (with `runner` now being the seventh positional argument).

- #215: (also #213, #214) Tweak tests & configuration sections of the code to include Windows compatibility. Thanks to Paul Moore.
- #212: Implement basic linting support using `flake8`, and apply formatting changes to satisfy said linting. As part of this shakeup, also changed all old-style (`%s`) string formatting to new-style (`{0}`). Thanks to Collin Anderson for the foundational patch.
- : Reorganize `Runner`, `Local` and `invoke.runner.run` for improved distribution of responsibilities & downstream subclassing.

> **Warning:** This includes backwards incompatible changes to the API signature of most members of the `invoke.runner` module, including `invoke.runner.run`. (However, in the case of `invoke.runner.run`, the changes are mostly in the later, optional keyword arguments.)

- #136: Added the `autoprint` flag to `invoke.tasks.Task`/`@task`, allowing users to set up tasks which act as both subroutines & "print a result" CLI tasks. Thanks to Matthias Lehmann for the original patch.
- : Fixed a sub-case of the already-mostly-fixed #149 so the error message works usefully even with no explicit collection name given.
- #162: Adjust platform-sensitive imports so Windows users don't encounter import-time exceptions. Thanks to Paul Moore for the patch.
- #119: (also #162, #113) Better handle platform-sensitive operations such as pty size detection or use, either replacing with platform-specific implementations or raising useful exceptions. Thanks to Gabi Davar and (especially) Paul Moore, for feedback & original versions of the final patchset.
- #167: Running the same task multiple times in one CLI session was horribly broken; it works now. Thanks to Erich Heine for the report.
- #165: Running `inv[oke]` with no task names on a collection containing a default task should (intuitively) have run that default task, but instead did nothing. This has been fixed.
- #169: Overhaul the Sphinx docs into two trees, one for main project info and one for versioned API docs.
- #142: The refactored Loader class failed to account for the behavior of `imp.find_module` when run against packages (vs modules) and was exploding at load time. This has been fixed. Thanks to David Baumgold for catch & patch.
- #145: Ensure a useful message is displayed (instead of a confusing exception) when listing empty task collections.
- #149: Print a useful message to stderr when Invoke can't find the requested collection/tasks file, instead of displaying a traceback.
- #140: Revert incorrect changes to our `setup.py` regarding detection of sub-packages such as the vendor tree & the parser. Also add additional scripting to our Travis-CI config to catch this class of error in future. Thanks to Steven Loria and James Cox for the reports.
- #125: Improve output of Failure exceptions when printed.

- #124: Add a `--debug` flag to the core parser to enable easier debugging (on top of existing `INVOKE_DEBUG` env var.)
- #87: (also #92) Rework the loader module such that recursive filesystem searching is implemented, and is used instead of searching `sys.path`.

  This adds the behavior most users expect or are familiar with from Fabric 1 or similar tools; and it avoids nasty surprise collisions with other installed packages containing files named `tasks.py`.

  Thanks to Michael Hahn for the original report & PR, and to Matt Iversen for providing the discovery algorithm used in the final version of this change.

  Warning: This is technically a backwards incompatible change (reminder: we're not at 1.0 yet!). You'll only notice if you were relying on adding your tasks module to `sys.path` and then calling Invoke elsewhere on the filesystem.

- #110: Add task docstrings' 1st lines to `--list` output. Thanks to Hiroki Kiyohara for the original PR (with assists from Robert Read and James Thigpen.)
- #115: Make it easier to reuse Invoke's primary CLI machinery in other (non-Invoke-distributed) bin-scripts. Thanks to Noah Kantrowitz.
- #135: (also bugs #120, #123) Implement post-tasks to match pre-tasks, and allow control over the arguments passed to both (via `invoke.tasks.call`). For details, see Pre- and post-tasks.

  Warning: Pre-tasks were overhauled a moderate amount to implement this feature; they now require references to **task objects** instead of **task names**. This is a backwards incompatible change.

- #127: Fill in tasks' exposed `name` attribute with body name if explicit name not given.
- #116: Ensure nested config overrides play nicely with default tasks and pre-tasks.
- #131: Make sure one's local tasks module is always first in `sys.path`, even if its parent directory was already somewhere else in `sys.path`. This ensures that local tasks modules never become hidden by third-party ones. Thanks to `@crccheck` for the early report and to Dorian Puła for assistance fixing.
- #121: Add missing help output denoting inverse Boolean options (i.e. `--[no-]foo` for a `--foo` flag whose value defaults to true.) Thanks to Andrew Roberts for catch & patch.
- #128: Positional arguments containing underscores were not exporting to the parser correctly; this has been fixed. Thanks to J. Javier Maestro for catch & patch.
- : Refactor the `invoke.runners.Runner` module to differentiate what it means to run a command in the abstract, from execution specifics. Top level API is unaffected.
- #117: Tidy up `setup.py` a bit, including axing the (broken) `distutils` support. Thanks to Matt Iversen for the original PR & followup discussion.
- #118: Update the bundled `six` plus other minor tweaks to support files. Thanks to Matt Iversen.
- #25: Trim a bunch of time off the test suite by using mocking and other tools instead of dogfooding a bunch of subprocess spawns.
- #107: Update configuration merging behavior for more flexible reuse of imported task modules, such as parameterizing multiple copies of a module within a task tree.
- #108: Update `invoke.collection.Collection.from_module` to accept useful shorthand arguments for tweaking the `invoke.collection.Collection` objects it creates (e.g. name, configuration.)

- #109: Add a `default` kwarg to `invoke.collection.Collection.add_task` allowing per-collection control over default tasks.
- #98: **BACKWARDS INCOMPATIBLE CHANGE!** Configuration merging has been reversed so outer collections' config settings override inner collections. This makes distributing reusable modules significantly less silly.
- #96: Tasks in subcollections which set explicit names (via e.g. `@task(name='foo')`) were not having those names honored. This is fixed. Thanks to Omer Katz for the report.
- #89: Implemented configuration for distributed task modules: can set config options in `invoke.collection.Collection` objects and they are made available to contextualized tasks.
- #86: Task arguments named with an underscore broke the help feature; this is now fixed. Thanks to Stéphane Klein for the catch.
- #83: Fix a bug preventing underscored keyword arguments from working correctly as CLI flags (e.g. `mytask --my-arg` would not map back correctly to `mytask(my_arg=...)`.) Credit: `@akitada`.
- #81: Fall back to sane defaults for PTY sizes when autodetection gives insane results. Thanks to `@akitada` for the patch.
- #57: Optional-value flags added - e.g. `--foo` tells the parser to set the `foo` option value to True; `--foo myval` sets the value to "myval". The built-in `--help` option now leverages this feature for per-task help (e.g. `--help` displays global help, `--help mytask` displays help for `mytask` only.)
- #55: A bug in our vendored copy of `pexpect` clashed with a Python 2->3 change in import behavior to prevent Invoke from running on Python 3 unless the `six` module was installed in one's environment. This was fixed - our vendored `pexpect` now always loads its sibling vendored `six` correctly.

1.2 Frequently asked questions

1.2.1 General project questions

Why was Invoke split off from the Fabric project?

Fabric (1.x and earlier) was a hybrid project implementing two feature sets: task execution (organization of task functions, execution of them via CLI, and local shell commands) and high level SSH actions (organization of servers/hosts, remote shell commands, and file transfer).

For use cases requiring both feature sets, this arrangement worked well. However, over time it became clear many users only needed one or the other, with local-only users resenting heavy SSH/crypto install requirements, and remote-focused users struggling with API limitations caused by the hybrid codebase.

When planning Fabric 2.x, having the "local" feature set as a standalone library made sense, and it seemed plausible to design the SSH component as a separate layer above. Thus, Invoke was created to focus exclusively on local and abstract concerns, leaving Fabric 2.x concerned only with servers and network commands.

Fabric 2 leverages many parts of Invoke's API, and allows (but does not require!) use of Invoke's CLI features, allowing multiple use cases (build tool, high level SSH lib, hybrid build/orchestration tool) to coexist without negatively impacting each other.

1.2.2 Defining/executing tasks

My task's first argument isn't showing up in `--help`!

This problem pops up if you forget to define an initial context argument for your task.

For example, can you spot the problem in this sample task file?

```
from invoke import task

@task
def build(c, where, clean=False):
    pass

@task
def clean(what):
    pass
```

This task file doesn't cause obvious errors when sanity-checking it with `inv --list` or `inv --help`. However, `clean` forgot to set aside its first argument for the context - so Invoke is treating `what` as the context argument! This means it doesn't show up in help output or other command-line parsing stages.

The command line says my task's first argument is invalid!

See *My task's first argument isn't showing up in –help!* - it's probably the same issue.

1.2.3 Running local shell commands (`run`)

Why is my command behaving differently under Invoke versus being run by hand?

99% of the time, adding `pty=True` to your `run` call will make things work as you were expecting. Read on for why this is (and why `pty=True` is not the default).

Command-line programs often change behavior depending on whether a controlling terminal is present; a common example is the use or disuse of colored output. When the recipient of your output is a human at a terminal, you may want to use color, tailor line length to match terminal width, etc.

Conversely, when your output is being sent to another program (shell pipe, CI server, file, etc) color escape codes and other terminal-specific behaviors can result in unwanted garbage.

Invoke's use cases span both of the above - sometimes you only want data displayed directly, sometimes you only want to capture it as a string; often you want both. Because of this, there is no "correct" default behavior re: use of a pseudo-terminal - some large chunk of use cases will be inconvenienced either way.

For use cases which don't care, direct invocation without a pseudo-terminal is faster & cleaner, so it is the default.

Calling Python or Python scripts prints all the output at the end of the run!

Note: This is typically a problem under Python 3 only.

The symptom is easy to spot - you're running a command that takes a few seconds or more to execute, it usually prints lines of text as it goes, but via `run` nothing appears to happen at first, and then all the output prints once it's done executing.

This is usually due to Python - the "inner" Python executable you're invoking, not the one Invoke is running under - performing unwanted buffering of its output streams. It does this when it thinks it's being called in a non-interactive fashion.

The fix is to force Invoke to run the command in a pseudoterminal by saying `pty=True` (e.g. `run("python foo", pty=True)`).

Alternately, since both Invoke and the inner command are Python, you could try loading the inner Python module directly in your Invoke-using code, and call whichever methods its command-line stub is using - instead of using `run`. This can often have other benefits too.

Why do I sometimes see `err: stdin: is not a tty`?

See *Why is my command behaving differently under Invoke versus being run by hand?* - the same root cause (lack of a PTY by default) is probably what's going on. In some cases (such as via the Fabric library) it's happening because a shell's login files are calling programs that require a PTY (e.g. `biff` or `mesg`) so make sure to look there if the actual foreground command doesn't seem at fault.

Everything just exits silently after I run a command!

Double check the command's exit code! By default, receiving nonzero exit codes at the end of a `run` call will result in Invoke halting execution & exiting with that same code. Some programs (pylint, Nagios check scripts, etc) use exit codes to indicate non-fatal status, which can be confusing.

The solution here is to add `warn=True` to your `run` call, which disables the automatic exit behavior. Then you can check the result's `.exited` attribute by hand to determine if it truly succeeded.

The auto-responder functionality isn't working for my password prompts!

Some programs write password prompts or other output *directly* to the local terminal (the operating-system-level TTY device), bypassing the usual stdout/stderr streams. For example, this is exactly what `the stdlib's getpass module` does, if you're calling a program that happens to be written in Python.

When this happens, we're powerless, because all we get to see is the subprocess' regular output streams. Thankfully, the solution is usually easy: just add `pty=True` to your `run` call. Forcing use of an explicit pseudo-terminal usually tricks these kinds of programs into writing prompts to stderr.

I'm getting `IOError: Inappropriate ioctl for device` when I run commands!

This error typically means some code in your project or its dependencies has replaced one of the process streams (`sys.stdin`, `sys.stdout` or `sys.stderr`) with an object that isn't actually hooked up to a terminal, but which pretends that it is. For example, test runners or build systems often do this.

99% of the time, this pops up for stdin only, in which case you may be able to work around it by specifying `in_stream=False` to `run` (note: `False`, **not** `None`!)

Gory details

Technically, what's happened is that the object handed to Invoke's command executor as e.g. `run('command', in_stream=xxx)` (or `out_stream` or etc; and these all default to the `sys` members listed above) implements a `fileno` method that is not returning the ID of a real terminal file descriptor. Breaking the contract in this way is what's leading Invoke to do things the OS doesn't like.

We're always trying to make this detection smarter; if upgrading to the latest version of Invoke doesn't fix the problem for you, please submit a bug report including details about the values and types of `sys.stdin/stdout/stderr`. Hopefully we'll find another heuristic we can use!

1.3 Installing

1.3.1 Basic installation

The recommended way to get Invoke is to **install the latest stable release** via pip:

```
$ pip install invoke
```

We currently support **Python 2.7** and **Python 3.4+**. Users still on Python 2.6 or older, or 3.3 or older, are urged to upgrade.

As long as you have a supported Python interpreter, **there are no other dependencies**. Invoke is pure-Python, and contains copies of its few dependencies within its source tree.

Note: See this blog post for background on our decision to vendorize dependencies.

See also:

Development for details on source control checkouts / unstable versions.

1.4 Development

1.4.1 Obtaining a source checkout

Our Git repository is maintained on Github at pyinvoke/invoke. Please follow their instructions for cloning (or forking, then cloning, which is best if you intend to contribute back) the repository there.

Once downloaded, install the repo itself + its development dependencies by running `pip install -r dev-requirements.txt`.

1.4.2 Submitting bug reports or patches

We follow contribution-guide.org for all of our development - please go there for details on submitting patches, which branch(es) to work out of, and so on. Our issue tracker is on our GitHub page.

1.4.3 Changelog location

Invoke's changelog lives in `sites/www/changelog.rst` and is formatted using the Releases Sphinx plugin.

1.4.4 Running management tasks

Invoke uses itself for project management and has a number of tasks you can see with `inv --list`. Some specific tasks of note:

- `test` and `integration`: Runs the primary and integration test suites, respectively. (Most of the time you can ignore `integration` - it's mostly for use by CI systems or once-in-a-while sanity checks locally.)
- `www` and `docs` (and their subtasks like `docs.browse`): Builds the WWW site and the API docs, respectively.

Another good resource is to skim our `.travis.yml` file for the commands it executes - if submissions don't pass all of those commands to some degree, they won't pass Travis' CI builds either!

1.5 Prior art

Why another task-running/subprocess-spawning Python library? As usual, the short answer is "there were already great 80-90% solutions out there, but none that fit our needs 100%." Specifically:

- **Multiple tasks at once** - almost no other Python command-line oriented libraries allow for invocations like:

  ```
  runner --core-opts task1 --task1-opts task2 --task2-opts
  ```

  and the few that do have half-baked implementations of the feature or are lacking in other ways.
- **Ability to mirror and capture subprocess output simultaneously** (in addition to everything flowing from that, like the ability to transparently auto-respond) - the standard library's `subprocess` can't do this and most other tools choose one or the other, or have other tradeoffs such as not supporting (or *only* supporting!) pseudoterminals.
- **Simplicity** - tools that try to do many things often suffer for it due to lack of focus. We wanted to build something clean and simple that just did one thing (ok. . . two things) well.
- **Customizability/control** - Invoke was designed to work well with (and be a foundation for) other tools such as Fabric's second version, and we felt that the work needed to adapt existing tools towards this goal would impede progress.

Some of the pre-existing solutions in this space in the Python world include:

- Argh: One of the more appealing options, but being built on argparse it doesn't support the multi-task invocation we needed. Also has its own "prior art" list which is worth your time.
- Baker: Nice and simple, but unfortunately too much so for our needs.
- Paver: Tries to do too much, clunky API, user-hostile error messages, multi-task feature existed but was lacking.
- Argparse: The modern gold standard for CLI parsing (albeit without command execution). Unfortunately, we were unable to get multiple tasks working despite lots of experimentation. Multiple tasks with their own potentially overlapping argument names, simply doesn't mesh with how `argparse` thinks about the command line.
- Click: is actually not pre-existing (Invoke's first public releases predate Click by a number of years) but it deserves mention anyway, as it's become popular in this particular niche.

1.6 Contact

You can get in touch with the developer & user community in any of the following ways:

- Bug reports and feature requests: first read contribution-guide.org, then check out our GitHub page.
- IRC: `#invoke` on Freenode
- Twitter: you've got a few options here:

  - @bitprophet is the canonical source for updates, but is also the developer's personal account (hint: you can turn off retweets and only see original content!)
  - @pyfabric is a much lower-traffic, announcement-only account that also serves the Fabric project; given how much Fabric is built directly on top of Invoke, many of the posts will be relevant to Invoke-only users.
  - @pyinvoke was set up for Invoke-specific announcements, but it only has a dozen followers so we've unfortunately let it languish. Should we automate our release process further, this account may get posts again, and we'll update this page accordingly.
- Blog: TK